Microsoft

Microsoft Developer Experience GitHub Copilot Demo

01/17/2024

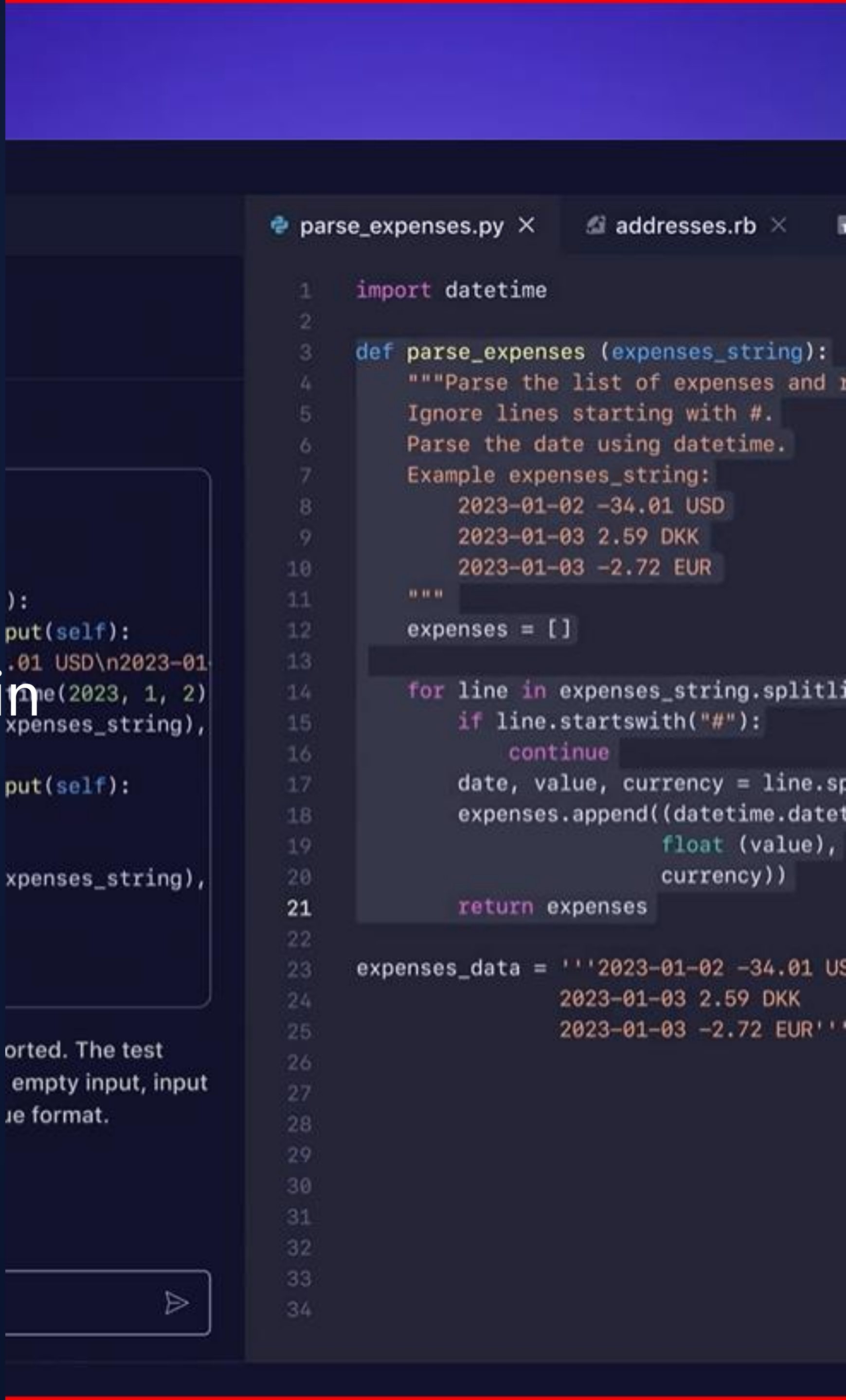
Gary.Ciampa@microsoft.com

Microsoft

N726R
FMS
MAP
AUX
122.90
5626
THROTTLE
PROPELLER
MIXTURE

GitHub CoPilot AI pair-programmer

- AI developer tool, helps write code faster with less work
- Draws context from comments & code to suggest code/functions
- Developers code faster, focus on solving problems, stay in context & feel more fulfilled with their work
- Powered by OpenAI Codex, generative pretrained language models
- Visual Studio Code, Visual Studio, Neovim, and the JetBrains suite of integrated development environments (IDEs).



```
):
put(self):
.01 USD\n2023-01
me(2023, 1, 2)
xpenses_string),

put(self):

xpenses_string),
```

orted. The test
empty input, input
ue format.

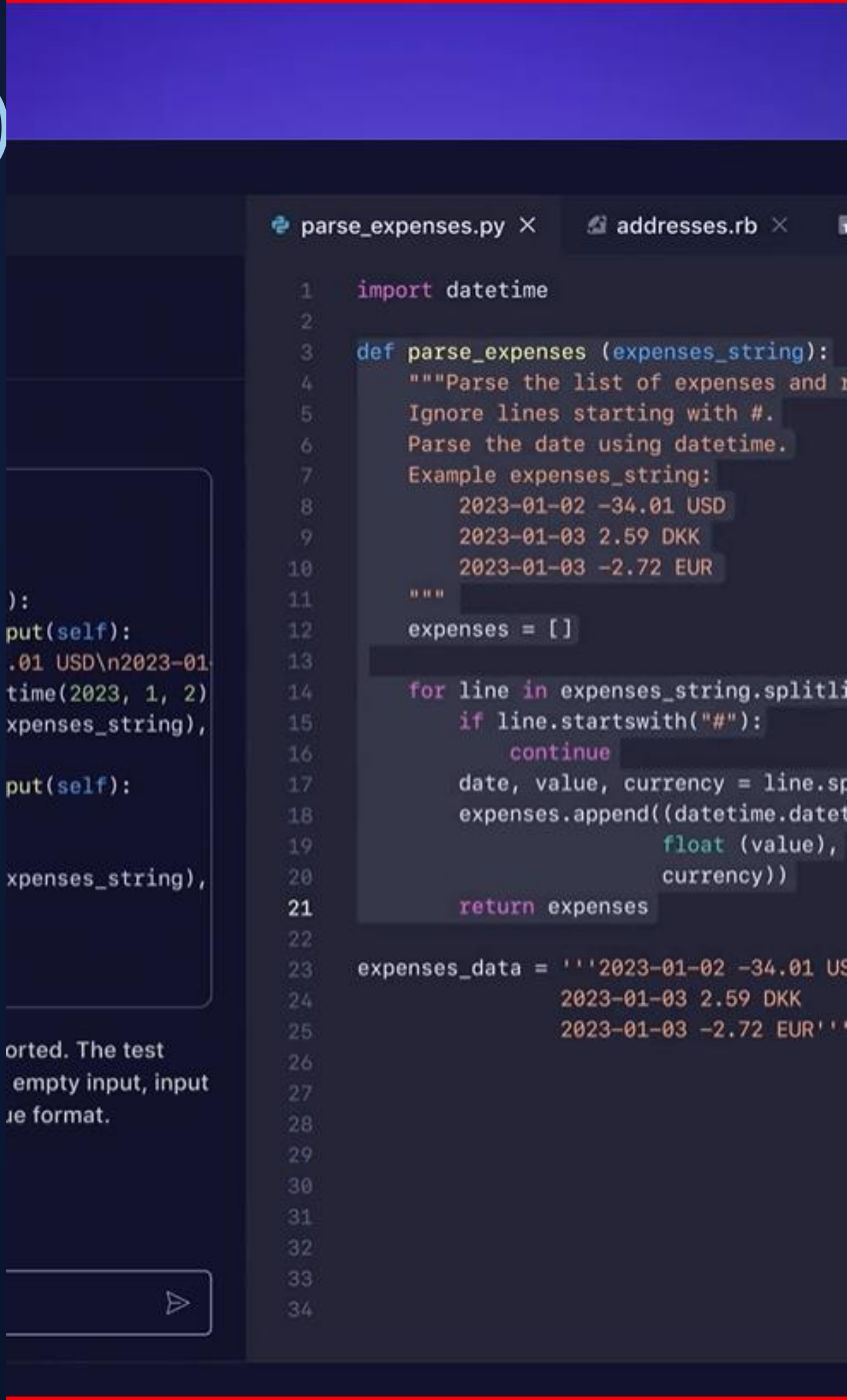
```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and r
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                        float (value),
                        currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 US
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

GitHub CoPilot prompt engineering (4s')

- **Single**: focus prompt on a single, well-defined task or question. Clarity is crucial for eliciting accurate and useful responses.

- **Specific**: instructions are explicit and detailed. Specificity leads to more applicable and precise code suggestions.

- **Short**: keep prompts concise and to the point, balance ensures clarity without overloading.
- .
- **Surround**: retain environment elements for tailored code suggestions, filenames, variables, methods



```
):
put(self):
.01 USD\n2023-01
time(2023, 1, 2)
xpenses_string),

put(self):


xpenses_string),
```

orted. The test
empty input, input
ue format.

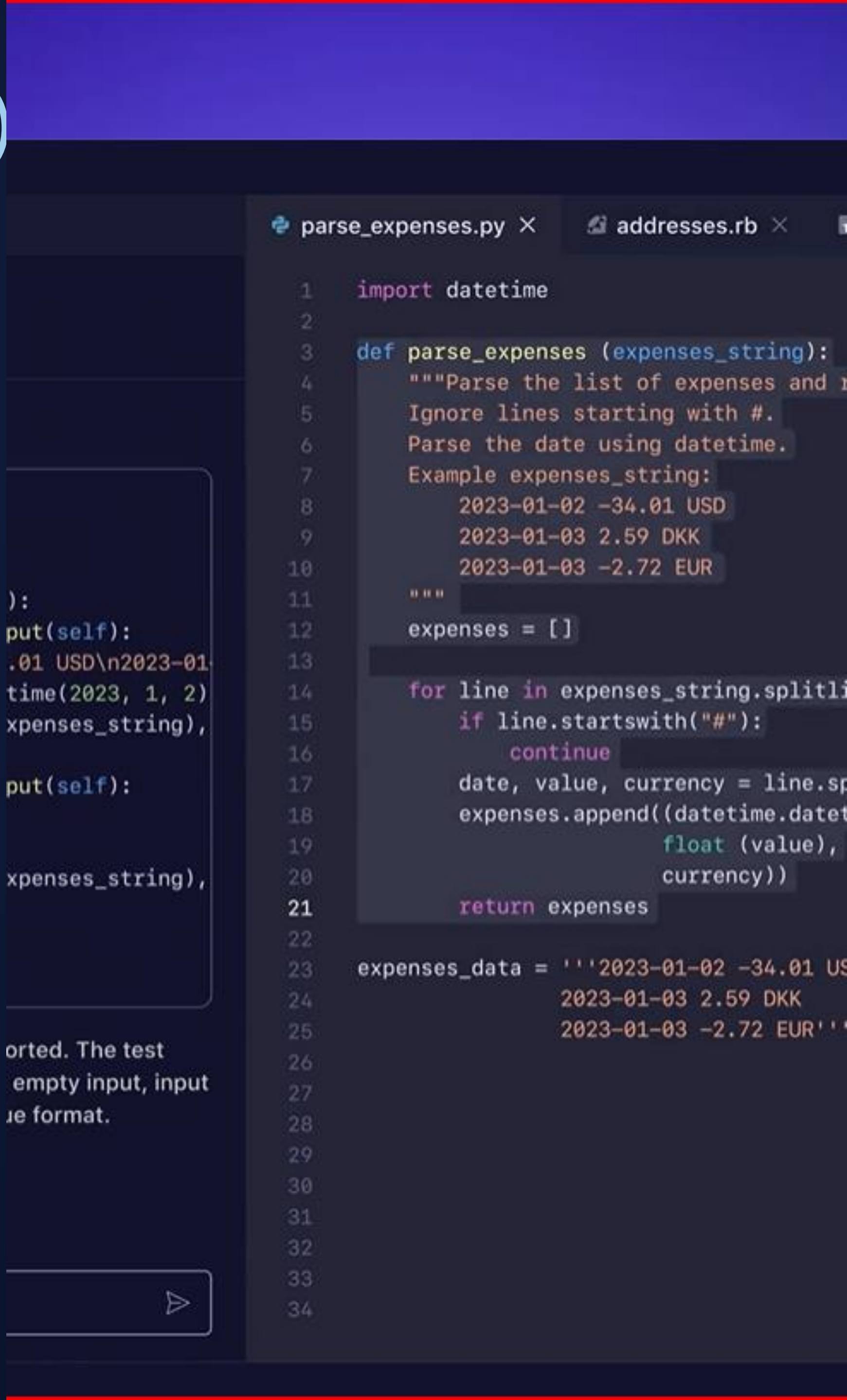
```python
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and r
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                        float (value),
                        currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 US
               2023-01-03 2.59 DKK
               2023-01-03 -2.72 EUR'''
```

GitHub CoPilot prompt engineering (4s’)

- **Single**: focus prompt on a single, well-defined task or question. Clarity is crucial for eliciting accurate and useful responses.

- **Specific**: instructions are explicit and detailed. Specificity leads to more applicable and precise code suggestions.

- **Short**: keep prompts concise and to the point, balance ensures clarity without overloading.
- .
- **Surround**: retain environment elements for tailored code suggestions, filenames, variables, methods



```
):
put(self):
.01 USD\n2023-01
time(2023, 1, 2)
xpenses_string),

put(self):

xpenses_string),
```

orted. The test
empty input, input
ıe format.

```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and r
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                        float (value),
                        currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 US
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

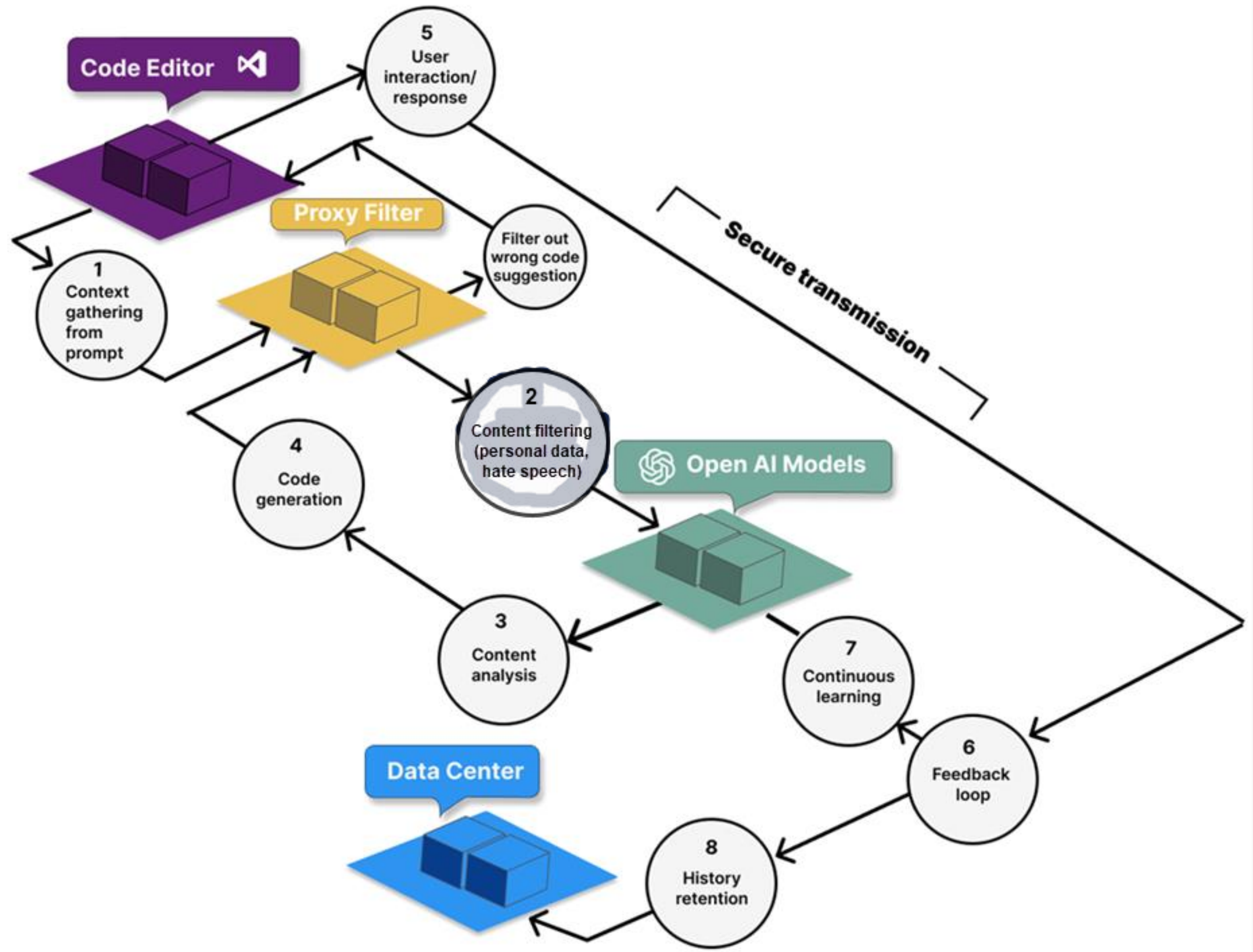
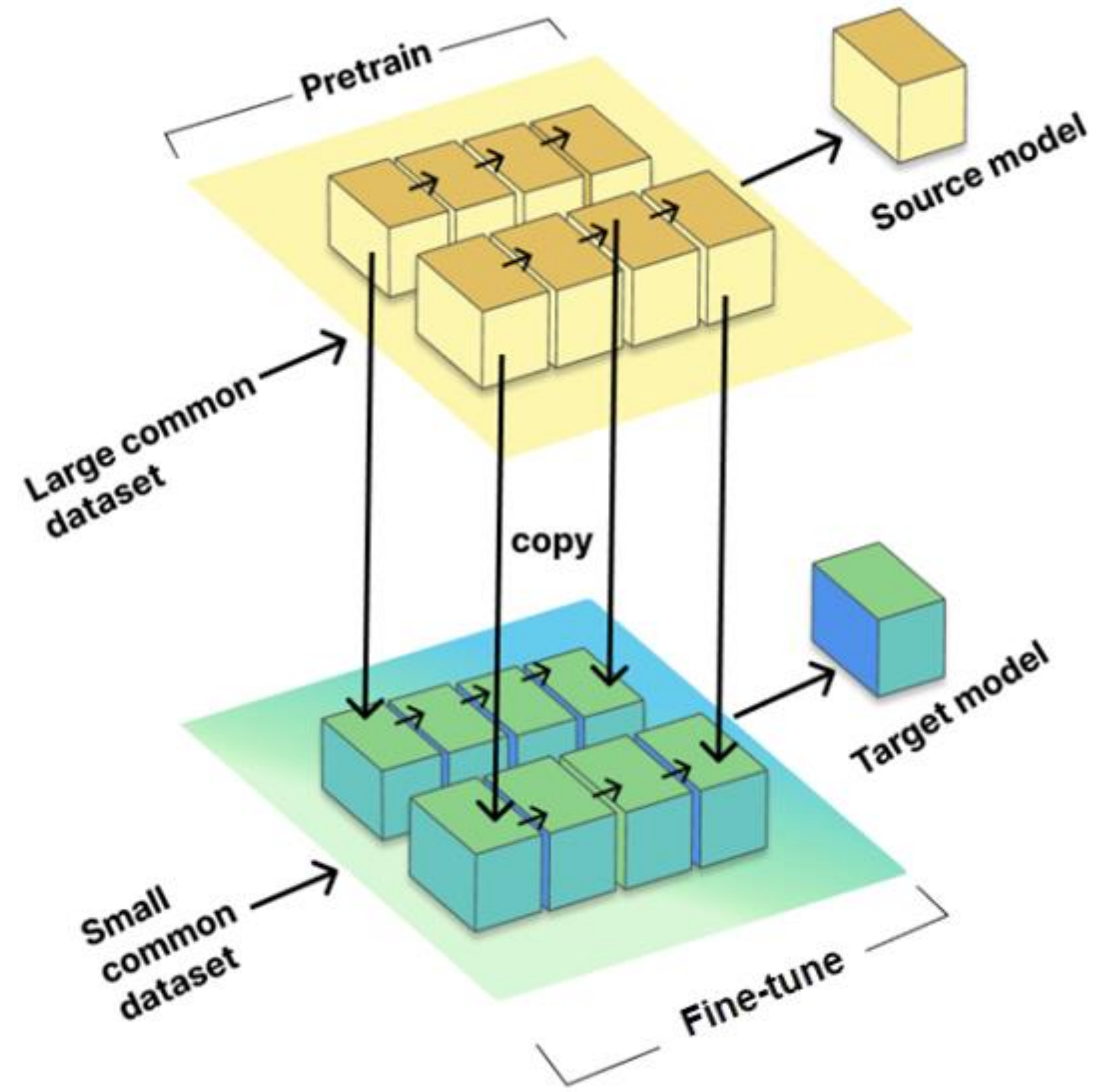


Illustration of Copilot Prompt Processing Flow

What is Fine-tuning?

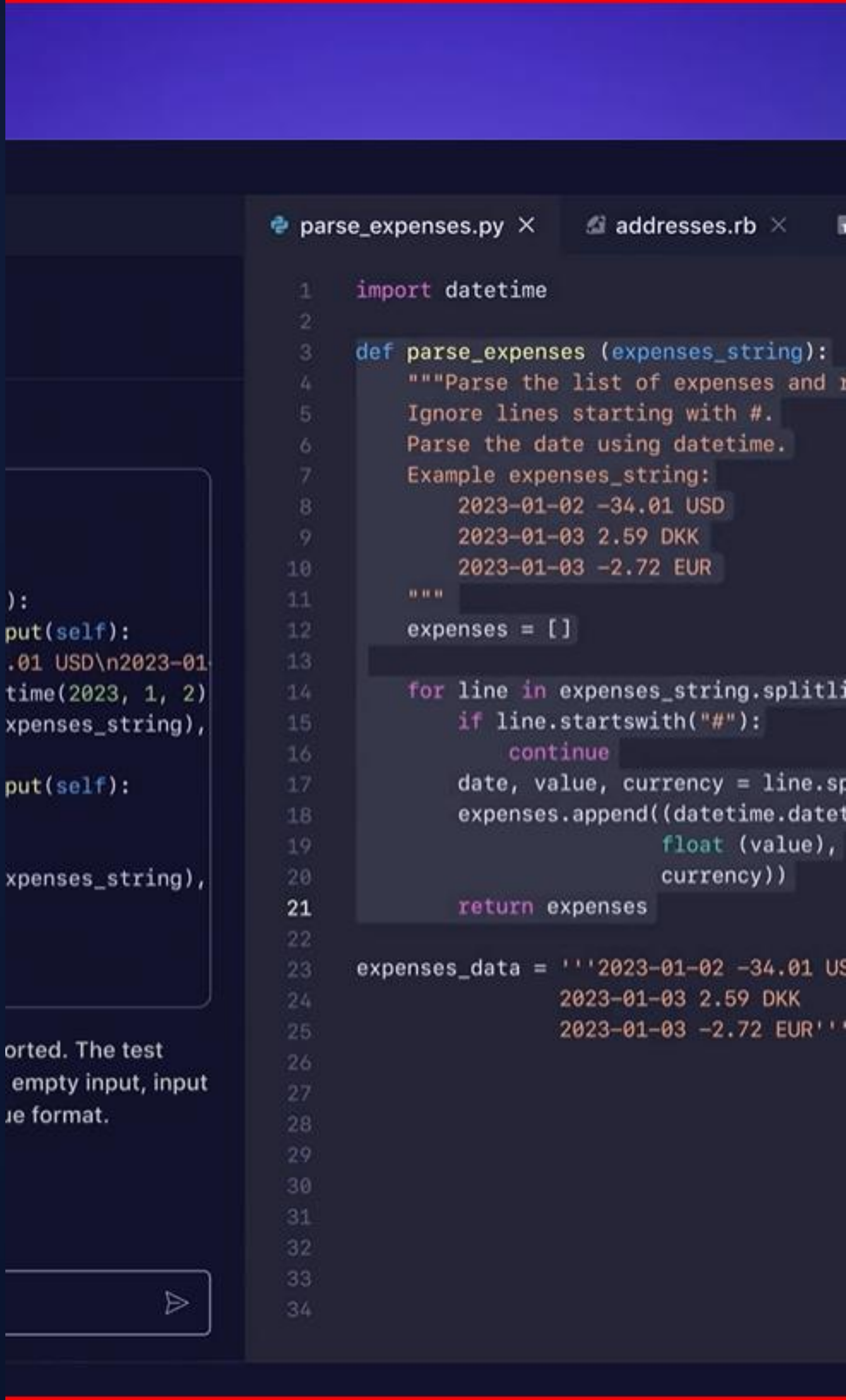


GitHub CoPilot Emphasis Items

- CoPilot is exactly that, co-pilot, not a replacement
- Training data, hints & suggestions
  - ➢ IOW: sometimes, recommendations are wrong
- Predictive, based on semantics & context

Developer is PIC

- Design patterns & structure
- Coding standards, source management & style
- Context & snippets must be vetted for expected output



```
):
put(self):
.01 USD\n2023-01
time(2023, 1, 2)
xpenses_string),

put(self):


xpenses_string),
```

orted. The test
empty input, input
ıe format.

```python
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                         float (value),
                         currency))
    return expenses

expenses_data = '''2023-01-02 -34.01 US
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR''
```

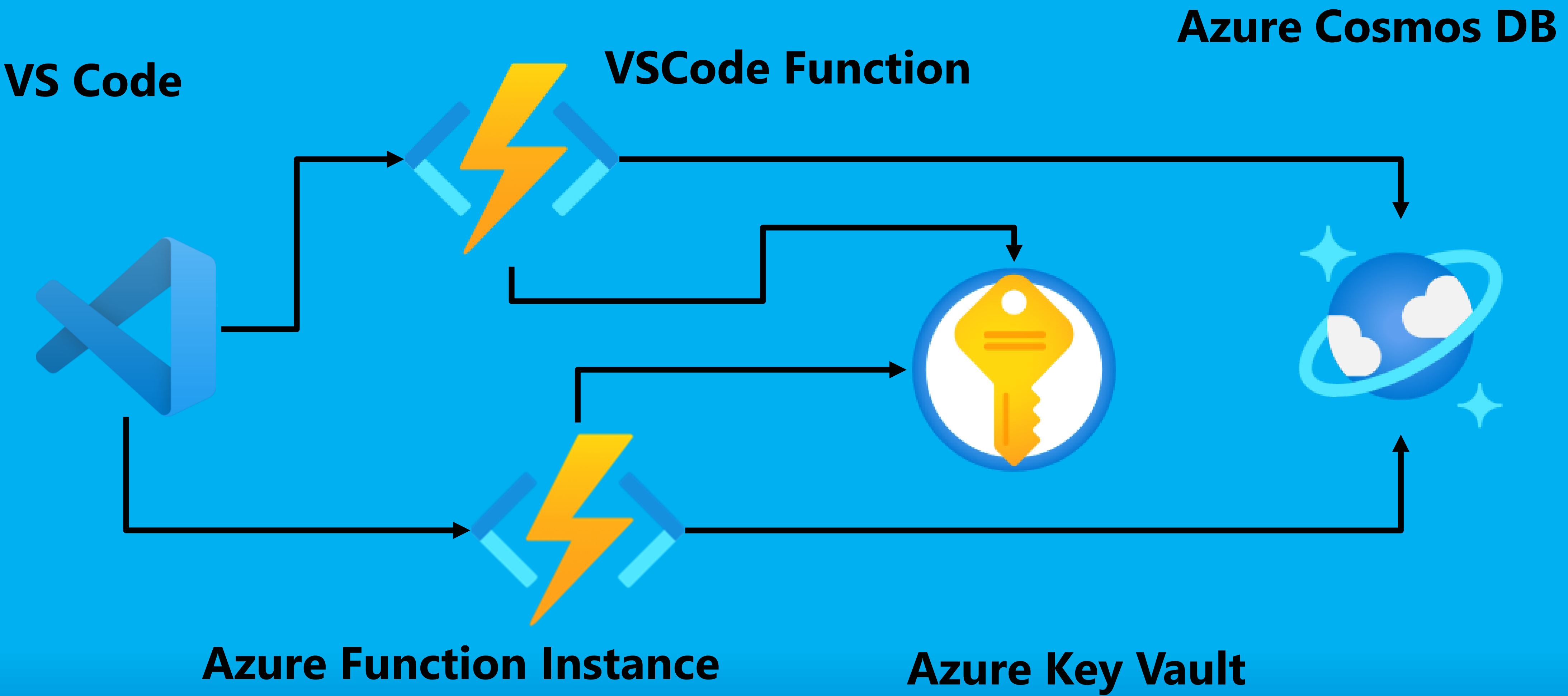
GitHub Copilot Scenario

1. Feature pulled from backlog for Sprint
2. Peer development team lacks required **time** & **experience** to implement feature
3. Director requested assistance to implement
4. Complete within 3 days & handoff to team for integration

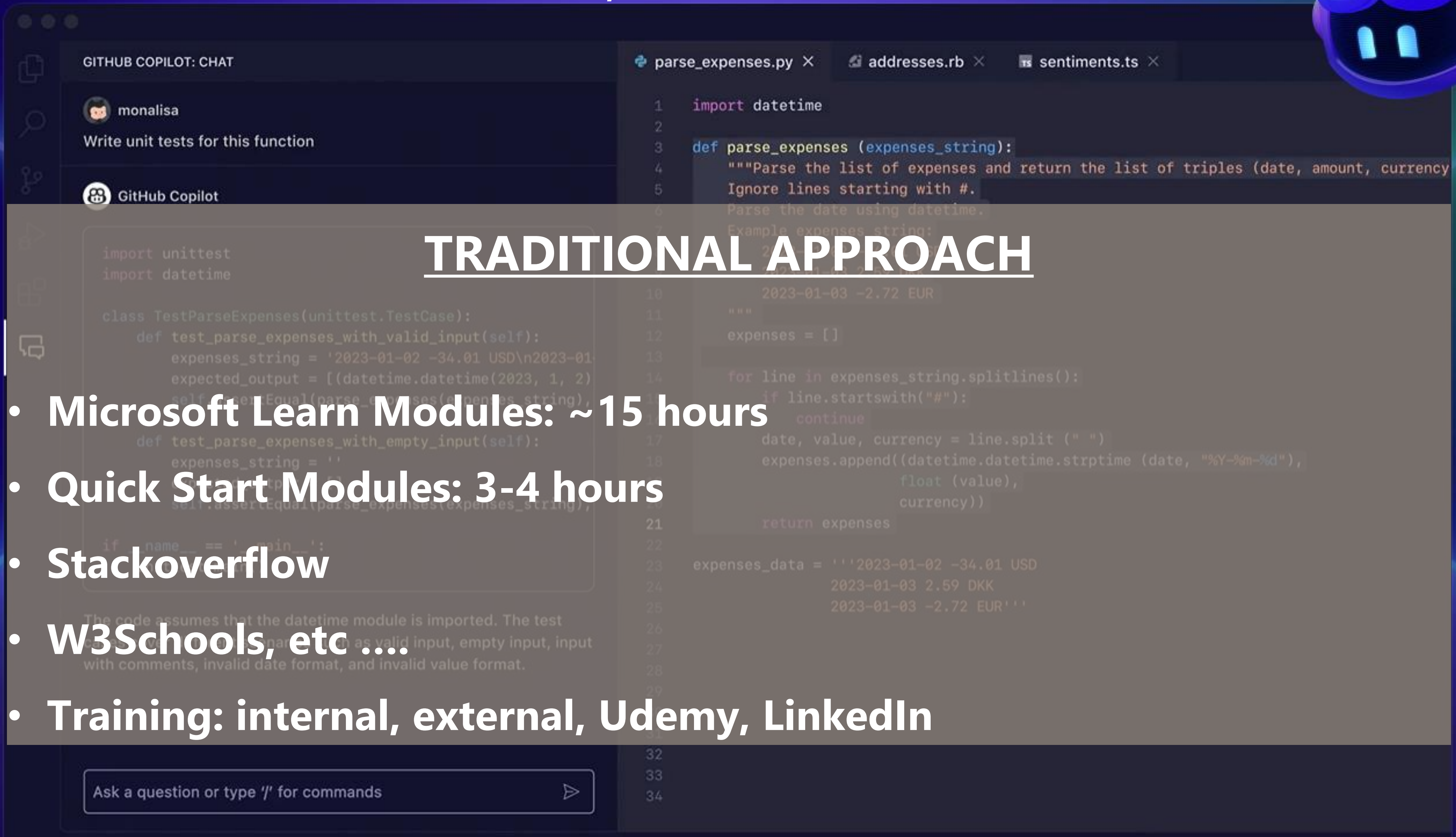
Feature requirements

1. Develop a compute service endpoint in Azure?
2. Requires a secure method to store secrets?
3. Persist data to Cosmos DB?
4. Service must be scalable to handle dynamic workloads?

Conceptual Architecture Diagram



Let's fail fast, VS Code & CoPilot



Write unit tests for this function

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):
    def test_parse_expenses_with_valid_input(self):
        expenses_string = '2023-01-02 -34.01 USD\n2023-01
        expected_output = [(datetime.datetime(2023, 1, 2)
        self.assertEqual(parse_expenses(expenses_string),

    def test_parse_expenses_with_empty_input(self):
        expenses_string = ''

        self.assertEqual(parse_expenses(expenses_string),

if __name__ == '__main__':
```

The code assumes that the datetime module is imported. The test [illegible] as valid input, empty input, input with comments, invalid date format, and invalid value format.

```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and return the list of triples (date, amount, currency
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split (" ")
        expenses.append((datetime.datetime.strptime (date, "%Y-%m-%d"),
                         float (value),
                         currency))
    return expenses

expenses_data = '''2023-01-02 -34.01 USD
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

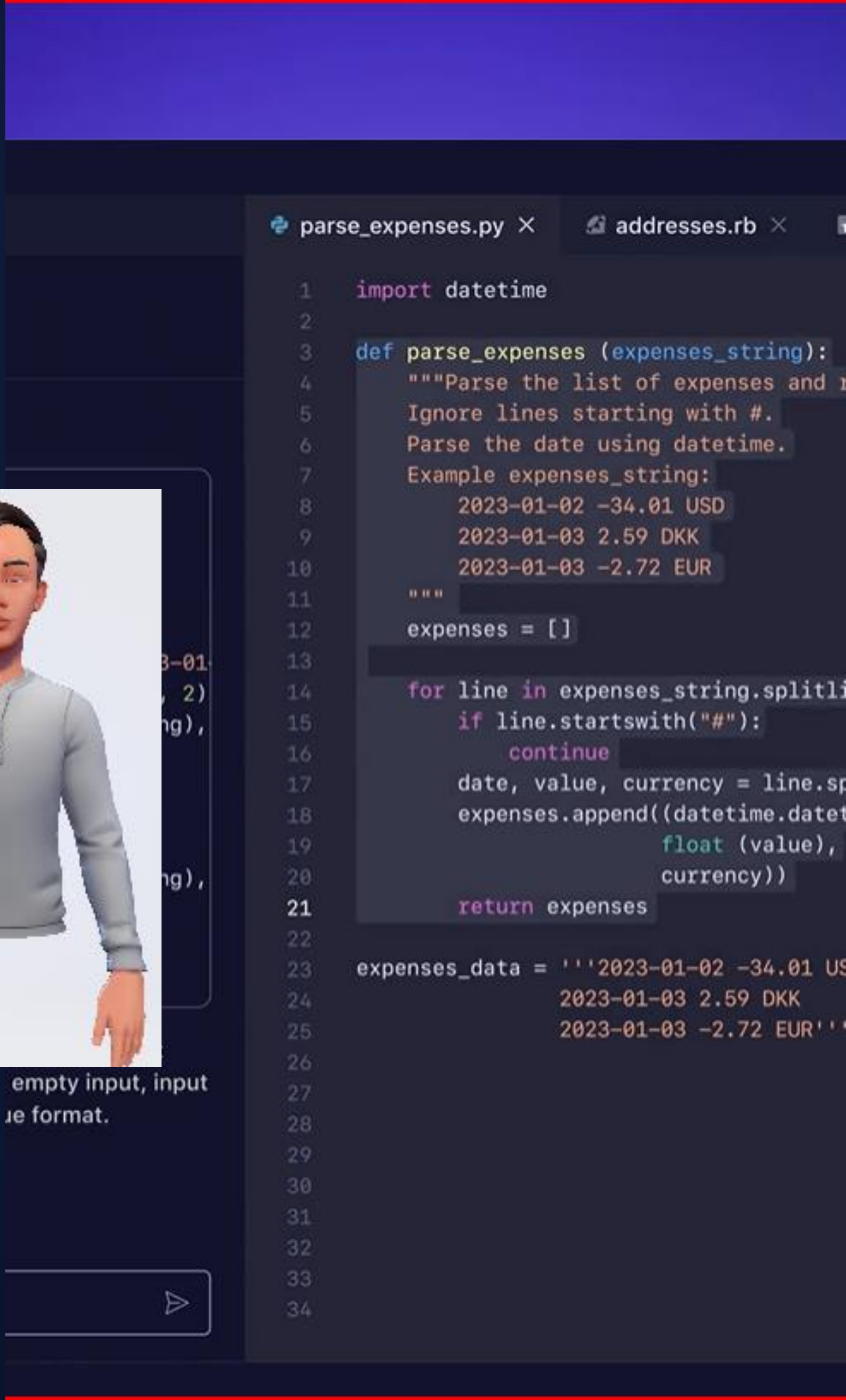
TRADITIONAL APPROACH

- **Microsoft Learn Modules: ~15 hours**
- **Quick Start Modules: 3-4 hours**
- **Stackoverflow**
- **W3Schools, etc ….**
- **Training: internal, external, Udemy, LinkedIn**

Quick start questions, considerations

- Azure Service: [Azure compute decision tree]?
- Visual Studio Code Extensions?
- .NET packages required?
- Prototype, scaling, REST endpoint?
- Securely manage, access a secret?
- Store persistent data?
- Challenges
  - ✓ Learning curve & .NET language
  - ✓ KeyVault URI & Secret
    - ➢ secret squirrel nut stash
  - ✓ CosmosDB & Container name

Virtual Joel



empty input, input
ue format.

parse_expenses.py × addresses.rb ×

```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and r
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                        float (value),
                        currency))
    return expenses

expenses_data = '''2023-01-02 -34.01 US
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

GitHub CoPilot references

GitHub CoPilot landing page

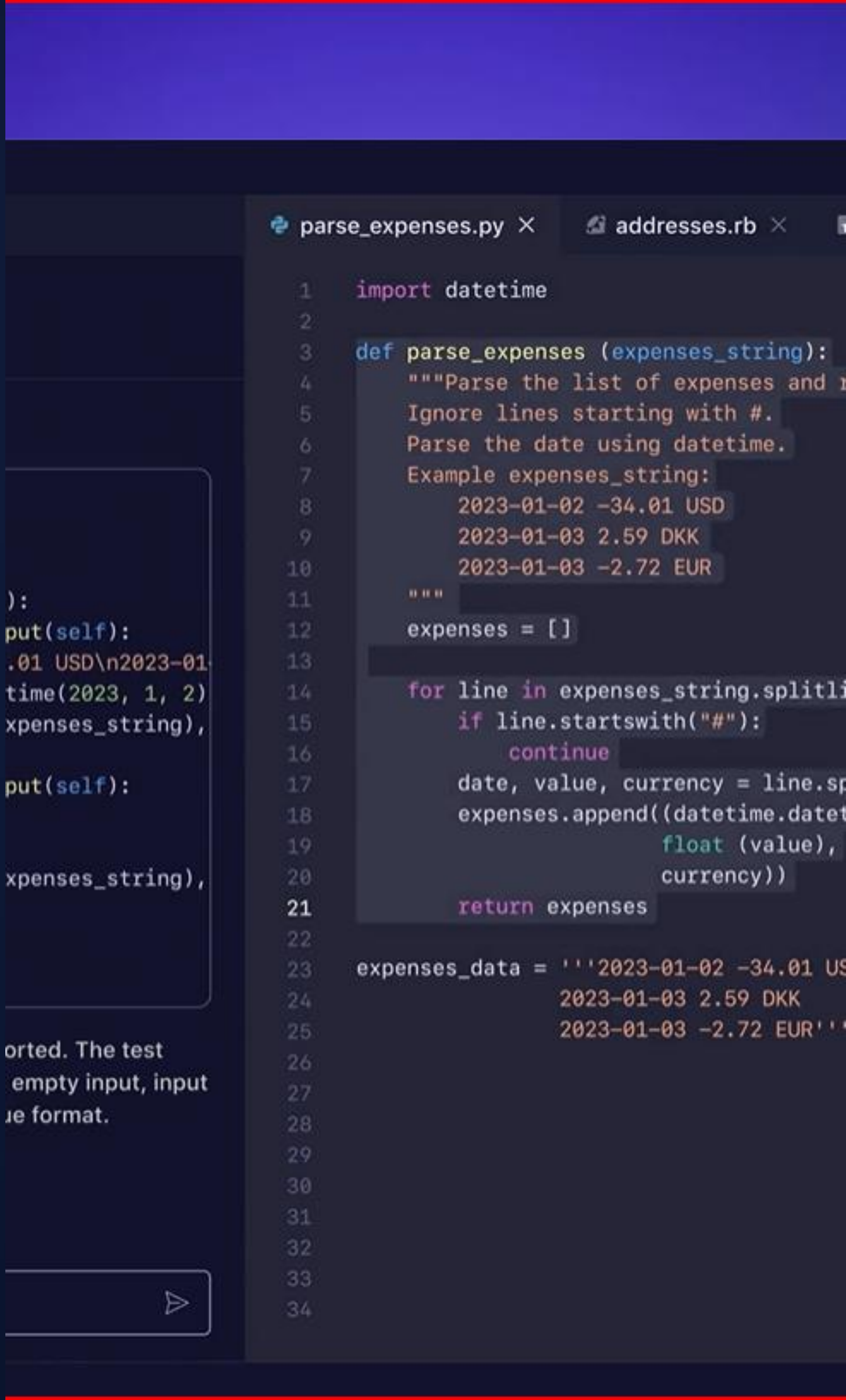
GitHub CoPilot Trust Center

GitHub CoPilot LinkedIn Tips & Tricks

GitHub CoPilot MSFT Build: May, 2023 BRK255H

GitHub CoPilot MSFT Learning Path

Visual Studio Code YouTube channel



```
):
put(self):
.01 USD\n2023-01
time(2023, 1, 2)
xpenses_string),

put(self):


xpenses_string),
```

orted. The test
empty input, input
ue format.

```python
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and r
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitli
        if line.startswith("#"):
            continue
        date, value, currency = line.sp
        expenses.append((datetime.datet
                        float (value),
                        currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 US
                2023-01-03 2.59 DKK
                2023-01-03 -2.72 EUR'''
```

Microsoft